Anytime Whole-Body Planning/Replanning for Humanoid Robots

Paolo Ferrari, Marco Cognetti, Giuseppe Oriolo

Anytime Whole-Body Planning/Replanning for Humanoid Robots

Paolo Ferrari[1], Marco Cognetti[2], Giuseppe Oriolo[1]

***Abstract*— In this paper we propose an anytime planning/replanning algorithm aimed at generating motions allowing a humanoid to fulfill an assigned task that implicitly requires stepping. The algorithm interleaves planning and execution intervals: a previously planned whole-body motion is executed while simultaneously planning a new solution for the subsequent execution interval. At each planning interval, a specifically designed randomized local planner builds a tree in configuration-time space by concatenating successions of CoM movement primitives. Such a planner works in two stages. A first lazy stage quickly expands the tree, testing only vertexes for collisions; then, a second validation stage searches the tree for feasible, collision-free whole-body motions realizing a solution to be executed during the next planning interval. We discuss how the proposed planner can avoid deadlock and we propose how it can be extended to a sensor-based planner. The proposed method has been implemented in V-REP for the NAO humanoid and successfully tested in various scenarios of increasing complexity.**

I. Introduction

The long-term challenge for humanoid robots is to substitute humans in repetitive and dangerous tasks. While in the past the researchers focused on obtaining robust and efficient locomotion, the real challenge nowadays is to generate whole-body motions aimed at fulfilling complex tasks.

Motion planning for humanoids is very challenging due to their characteristics: they have many degrees of freedom, resulting in a high-dimensional planning space; they are not free-flying systems, then several kinematic/dynamic constraints must be taken into account while generating feasible motions; finally, they must maintain equilibrium at all times. To make tractable the planning problem, many methods that rely on simplifying assumptions on either the environment or the robot geometry have been proposed (e.g., [1], [2], [3]).

The difficulties further increase when planning motions for a humanoid that is assigned a task, as discussed in [4], [5], [6]. Differently from these works, in [7], we introduced an approach that solves the task-oriented planning problem without separating locomotion from task execution. This planner is hinged on the concept of CoM movement primitives, defined as precomputed trajectories of the CoM that are associated to specific actions. We also used a similar approach in [8] for dealing with deformable tasks and in [9] for achieving natural reaching motions.

For complex planning problems, computing a complete solution (possibly considering its optimality), may be computationally expensive, or even impossible when planning under time limitations. To address such situation, many proposed methods rely on the idea of quickly computing an initial sub-optimal solution, which is improved until the available planning time runs out. Such methods are known as *anytime* algorithms. Firstly, the focus was to develop anytime versions of heuristic-based searches, like the ARA* algorithm [10]. Such method has been extended to dynamic contexts [11] and used within a footstep planning framework [12]. Although the latter finds efficient footstep plans in short planning times, it does not aim at computing feasible whole-body motions allowing the robot to achieve an assigned task. Other works ([13], [14]) propose anytime versions of sampling-based algorithms, particularly suited for planning in high-dimensional, non-uniform cost spaces. So far, their effectiveness has been shown only for wheeled vehicles.

In this work, we propose an anytime planning/replanning algorithm aimed at generating motions allowing a humanoid to fulfill an assigned task that implicitly requires stepping. Our algorithm interleaves planning and execution intervals: a previously planned whole-body motion is executed while simultaneously planning a new solution for the subsequent execution interval. At each planning interval, differently from the classical anytime paradigm, our algorithm focuses on obtaining the best solution, guaranteed to be valid in the next execution interval, among those that the planner has been able to produce within the deliberation time, without relying on continuous refinements of the initial solution. The overall humanoid motion results to be constituted by a sequence of on-line computed short horizon solutions.

Typically, motion planners running time is mostly spent in collision checking. On the basis of such observation, many methods have been proposed that postpone collision checks until they are strictly required. Such methods are known as *lazy* algorithms. In [15], a probabilistic roadmap is initially constructed assuming that all nodes and edges are collision-free, and then repeatedly searched for shortest paths, checking collisions along them. A bi-directional PRM using lazy collision checking is introduced in [16], where the method is successfully applied for a team of manipulators working in an automotive body shop. The lazy approach has been also involved in asymptotically-optimal planners [17], while applications to humanoids are so far limited to situations in which the robot does not need to take steps [18].

Here, we present an anytime framework that makes use of a specifically designed randomized planner, invoked at each replanning interval, that builds a tree in configuration-time space by concatenating successions of CoM movement primitives. Such planner a works in two stages. A first lazy

[1] Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Universita di Roma, via Ariosto 25, 00185 Roma, Italy. E-mail: {ferrari, oriolo}@diag.uniroma1.it.

[2] CNRS, Univ Rennes, Inria, IRISA, Rennes, France, E-mail: marco.cognetti@irisa.fr.

This work is supported by the EU H2020 COMANOID project.

stage quickly expands the tree, testing only vertexes for collisions; then, a second validation stage searches the tree for feasible, collision-free whole-body motions realizing a solution to be executed during the next planning interval.

The rest of the paper is organized as follows. We introduce the planning problem in Sect. II. The anytime planning/replanning scheme is introduced in Sect. III, hinged on a local motion planner discussed in Sect. IV. V-REP planning experiments for the NAO humanoid robot are illustrated in Sect. V. Finally, Sect. VI describes an extension of the proposed algorithm for managing planning deadlocks, while Sect. VII gives some insights for the sensor-based extension of the algorithm. Conclusions end the paper in Sect. VIII.

II. Problem Formulation

Before formulating our motion planning problem, we recall the humanoid motion model introduced in [7], [8].

We define the configuration of the humanoid as

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{q}_{\mathrm{CoM}} \\ \boldsymbol{q}_{\mathrm{jnt}} \end{pmatrix},$$

where $\boldsymbol{q}_{\mathrm{CoM}} \in SE(3)$ is the world pose (position and orientation) of a reference frame attached to the Center of Mass (CoM) and $\boldsymbol{q}_{\mathrm{jnt}} \in \mathcal{C}_{\mathrm{jnt}}$ is the n-vector of joint angles.

Within our planner, the CoM movements will be generated by patching subtrajectories extracted by a precomputed catalogue of *CoM movement primitives*. These primitives represent elementary humanoid motions, such as crouching, stepping, and so on. Each primitive has a given duration and may specify trajectories for other points of the robot in addition to the CoM: for example, a stepping primitive will include a swing foot trajectory.

This planning approach is reflected in the hybrid (partly algebraic, partly differential) motion model

$$\boldsymbol{q}_{\mathrm{CoM}}(t) = \boldsymbol{q}_{\mathrm{CoM}}(t_k) + \boldsymbol{A}(\boldsymbol{q}_{\mathrm{CoM}}(t_k))\,\boldsymbol{u}_{\mathrm{CoM}}(t) \quad (1)$$

$$\dot{\boldsymbol{q}}_{\mathrm{jnt}}(t) = \boldsymbol{v}_{\mathrm{jnt}}(t), \quad (2)$$

where $t \in [t_k, t_k + T_k]$, with T_k the duration of the current CoM primitive. In this model, $\boldsymbol{A}(\boldsymbol{q}_{\mathrm{CoM}}(t_k))$ is the transformation matrix between the CoM frame at t_k and the world frame, $\boldsymbol{u}_{\mathrm{CoM}}(t)$ is the CoM pose displacement at t relative to t_k (as specified by the primitive), and $\boldsymbol{v}_{\mathrm{jnt}}(t)$ is the vector of joint velocity commands (which must obviously be compatible with the primitive itself).

In the situation of interest, the humanoid is assigned a loco-manipulation task, i.e., a manipulation task that implicitly requires locomotion. In particular, the robot must bring a specified hand (hence, the *end-effector*) to a desired set-point, e.g., for grasping an object that is in general outside the workspace of the humanoid at its initial configuration. Composite tasks, like mobile manipulation, are also admissible, since they can be simply specified through a sequence of desired set-points (e.g., for picking an object and moving it to another location), simply by iteratively applying the proposed method over the set-points sequence. Furthermore, we assume that the robot lies in an environment populated by fixed obstacles, whose geometry is known.

Computing a complete plan for such problems requires high planning times, and the robot is forced to wait for the solution before being able to start moving. In this work, we propose an anytime motion planner that allows the robot to start moving even if a complete plan is not yet retrieved, as future motions can be planned while moving.

Let $\boldsymbol{y}_M = \boldsymbol{f}(\boldsymbol{q})$ be the position variables of the chosen end-effector, and $\boldsymbol{y}_M^*$ the desired set-point. A solution to our motion planning problem consists of a whole-body motion $\boldsymbol{q}(t)$, $t \in [t_{\mathrm{ini}}, t_{\mathrm{fin}}]$, constituted by a sequence of on-line computed partial solutions, that satisfies four requirements:

- R1 The assigned set-point is reached at a finite time t_{fin} [1].
- R2 Collisions with workspace obstacles and self-collisions are avoided.
- R3 Position and velocity limits on the joints are respected.
- R4 The robot is in equilibrium at all times.

In the following, we will call *feasible* the motions that satisfy requirements R2-R4.

III. Anytime Planner/Replanner

To address the described problem, we propose an anytime planning/replanning algorithm that interleaves planning and execution intervals: a previously planned whole-body motion is executed while simultaneously planning a new solution for the subsequent execution interval.

During the i-th planning interval, the aim is to produce a whole-body motion that starts at the configuration $\bar{\boldsymbol{q}}_{i+1}$ that the robot will reach at the end of the simultaneously executed whole-body motion, and is guaranteed to be feasible within a limited region of the task space $\mathcal{S}(\bar{\boldsymbol{q}}_{i+1})$ (henceforth referred to as the *planning zone*), whose location and geometry depend on the configuration $\bar{\boldsymbol{q}}_{i+1}$. To illustrate the proposed method, we assume that the planning zone consists of a sphere of a given radius r (as depicted in Fig. 1) centered at the CoM position $\bar{\boldsymbol{p}}_{\mathrm{CoM},i+1}$ of the humanoid at the configuration $\bar{\boldsymbol{q}}_{i+1}$.

Hereafter, we will refer to the whole-body motion produced during a certain planning interval as *local plan*, as its feasibility is guaranteed only within a fixed planning zone.

The *i*-th planning interval consists in an invocation of a specifically designed local motion planner (LMP), described in the next section, that is allowed to run for a given time budget. While such time budget determines the duration of the *i*-th planning interval, the duration of the local plan to be computed is not specified, as it can be autonomously determined by the LMP. Consequently, the duration of the *i*+1-th execution interval is equal to the duration of the local plan found in the *i*-th planning interval. In the following, we denote the duration of, respectively, the *i*-th planning and execution intervals by $\Delta T_{P,i}$ and $\Delta T_{E,i}$.

The anytime planning/replanning algorithm is shown in Algorithm 1. It starts by invoking the LMP that is in charge of producing, within a given time budget $\overline{\Delta T}_P$, the first local plan, i.e., a whole-body motion $\boldsymbol{q}(t)$, $t \in [t_1, t_2 = t_1 + \Delta T_{E,1}]$, with $\Delta T_{E,1}$ its duration, to be performed

[1] In our formulation t_{fin} is not assigned but determined by the planner.

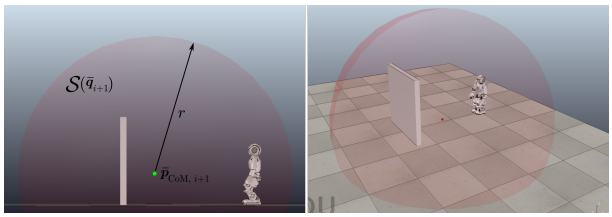

Fig. 1. An example of planning zone used in our framework.

Algorithm 1: Anytime Planner/Replanner

```
ΔT_{P,0} ← ΔT_P;
q(t), t ∈ [t_1, t_2] ← LMP(q_ini, ΔT_{P,0});
compute time budget ΔT_{P,1} according to (3);
extract q̄_2 from q(t);
i ← 1;
while f(q̄_{i+1}) ≠ y*_M do
    simultaneously do (1) and (2):
    (1) execute whole-body motion q(t) for t ∈ [t_i, t_{i+1}];
    (2) q(t), t ∈ [t_{i+1}, t_{i+2}] ← LMP(q̄_{i+1}, ΔT_{P,i});
    compute time budget ΔT_{P,i+1} according to (3);
    i ← i + 1;
    extract q̄_{i+1} from q(t);
end
execute whole-body motion q(t) for t ∈ [t_i, t_{i+1}];
```

within $\mathcal{S}(\boldsymbol{q}_{\text{ini}})$, with $\boldsymbol{q}_{\text{ini}}$ the initial robot configuration. Once such local plan is computed, our algorithm begins to perform execution and planning in parallel.

At the i-th generic iteration, the algorithm simultaneously:

- *executes the current local plan*, i.e., the whole-body motion $\boldsymbol{q}(t)$ valid in the time interval $[t_i, t_{i+1}]$ of duration $\Delta T_{E,i}$;
- *plans a novel local plan* within a given time budget $\Delta T_{P,i}$ (defined below in eq. (3)). This local plan starts at $\bar{\boldsymbol{q}}_{i+1} = \boldsymbol{q}(t_{i+1})$, i.e., the configuration that the robot will reach at the end of the current execution, and defines the motion within the next time interval $[t_{i+1}, t_{i+2} = t_{i+1} + \Delta T_{E,i+1}]$, where $\Delta T_{E,i+1}$ is the duration of the novel local plan.

When both procedures complete, the planning time budget $\Delta T_{P,i+1}$ for the next invocation of the LMP is assigned as

$$\Delta T_{P,i+1} = \Delta T_{E,i+1} \ . \tag{3}$$

where $\Delta T_{E,i+1}$ is the duration of the next execution interval.

This interleaved procedure terminates when the planner computes a local plan such that the desired set-point is definitely reached, i.e., $\boldsymbol{f}(\bar{\boldsymbol{q}}_{i+1}) = \boldsymbol{y}^*_M$ with $\boldsymbol{f}(\bar{\boldsymbol{q}}_{i+1})$ the end-effector position at the last configuration $\bar{\boldsymbol{q}}_{i+1}$ of the new local plan. In this case, the latter represents the last whole-body motion to be executed.

IV. Local Motion Planner (LMP)

In our anytime framework, whole-body motions to be executed in the subsequent time interval (at the beginning of which the robot will be in the configuration $\bar{\boldsymbol{q}}$) are generated by means of a CoM movement primitives-based local motion planner (LMP). This computes a local plan such that the robot is completely included in the current planning zone $\mathcal{S}(\bar{\boldsymbol{q}})$ while executing it. Building on ideas introduced in [7], [8], [9], the planner expands a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ in configuration-time space, where $\mathcal{V}$ and $\mathcal{E}$ are, respectively, sets of vertexes and edges. To this end, it uses a catalogue U of N *CoM movement primitives*. A typical U will include various stepping motions and possibly more complex movements, such as crouching, crawling, and so on. A fundamental element of U is `free_CoM`, a pure manipulation primitive. Except for the latter, all the primitives satisfy the R4 requirement (humanoid equilibrium) by construction. Static equilibrium is not guaranteed in `free_CoM` and must be explicitly checked.

A vertex $v = (\boldsymbol{q}, \boldsymbol{u}_{\text{CoM}}, \boldsymbol{w})$ consists of a configuration $\boldsymbol{q}$, with an associated time instant (not explicitly displayed), a CoM primitive $\boldsymbol{u}_{\text{CoM}}$, and a set of weights $\boldsymbol{w} = \{w^1, \ldots, w^N\}$. An edge represents a feasible whole-body motion that connects two adjacent vertexes.

The set $\boldsymbol{w}$ is used to assign a weight to each primitive in the catalogue, and acts as a probability distribution over the different possibilities of expanding the respective vertex. For each vertex, the weights $\boldsymbol{w}$ are initialized at predefined values $\boldsymbol{w}_{\text{init}}$ at the time of the vertex creation, and updated (see Sect. IV-A) each time the vertex is attempted to be expanded, for avoiding to generate in $\mathcal{T}$ already existing vertexes.

The planner makes use of a *compatibility* metric $d(\boldsymbol{q}, \bar{\boldsymbol{y}})$ that measures the compatibility of a configuration $\boldsymbol{q}$ with respect to a point $\bar{\boldsymbol{y}}$ in the task space. In particular, d is defined as the Euclidean distance between the ground projections of the robot CoM position at $\boldsymbol{q}$ and $\bar{\boldsymbol{y}}$.

The LMP works in two stages (see Procedure 1). The first stage implements a RRT-like strategy aimed at quickly expanding $\mathcal{T}$. To this end, it works in a *lazy* fashion, checking collisions with obstacles inside the zone $\mathcal{S}(\bar{\boldsymbol{q}})$ only at the vertexes level by means of a simplified robot model. At the end of this stage, a set of candidate local plans results from $\mathcal{T}$. In particular, a branch of the tree provides a *candidate local plan* if its ending vertex (i.e., the leaf) contains a configuration $\boldsymbol{q}$ such that one of the these conditions holds:

- The simplified robot model at the configuration $\boldsymbol{q}$ does not completely lie within the current planning zone.
- The configuration $\boldsymbol{q}$ is closer than a predefined threshold $\bar{d}$ (in terms of the described compatibility metric) to the desired set-point $\boldsymbol{y}^*_M$, i.e., $d(\boldsymbol{q}, \boldsymbol{y}^*_M) < \bar{d}$, as it potentially allows for completing the assigned task (through a `free_CoM` motion, see Sect. IV-B for details).

In the first case, the candidate local plan consists of the portion of the branch ending at the last vertex containing a configuration such that the simplified robot model completely lies within the planning zone. Instead, in the second case the candidate local plan consists of the entire branch.

The second stage is aimed at validating a candidate local plan resulting from the lazy stage, by generating the corresponding whole-body motions and checking collisions by means of the exact robot model, that allows either to safely proceed towards the destination or to fulfill the task.

Procedure 1: LMP($\bar{\boldsymbol{q}}$, ΔT_P)

split the time budget ΔT_P in two intervals ΔT_P^L and ΔT_P^V;
$\mathcal{T} \leftarrow$ LazyStage($\bar{\boldsymbol{q}}$, ΔT_P^L);
$\boldsymbol{q}(t) \leftarrow$ ValidationStage($\mathcal{T}$, ΔT_P^V);
return $\boldsymbol{q}(t)$;

The two planning stages are described in details in the following subsections. The LMP is allowed to run for a given time budget ΔT_P. Such time budget is split into two intervals, each one dedicated, respectively, to the lazy and the validation stages. When the deliberation time assigned to the lazy phase ΔT_P^L runs out, expansion of the tree is stopped. While, if the deliberation time assigned to the validation phase ΔT_P^V runs out before it returns a valid plan, a failure is reported. For sake of illustration, in the following we do not explicitly describe the interruption mechanism.

A. The Lazy Stage

The lazy stage (see Procedure 2) starts by rooting the tree $\mathcal{T}$ at $v_{\rm root} = (\bar{\boldsymbol{q}}, \bar{\boldsymbol{u}}_{\rm CoM}, \boldsymbol{w}_{\rm root})$, where $\bar{\boldsymbol{q}}$ is the configuration that the robot will reach at the end of the current execution interval, $\bar{\boldsymbol{u}}_{\rm CoM}$ is the CoM primitive through which it had been produced (such information may be extracted from $\mathcal{T}$ before re-rooting it), and $\boldsymbol{w}_{\rm root}$ is set to $\boldsymbol{w}_{\rm init}$. Initially the set of vertexes that can be selected for attempting to expand the tree $\mathcal{T}$ contains only the root vertex $v_{\rm root}$. Denote by $\mathcal{V}'$ such set, which consists in a subset of $\mathcal{V}$.

A generic iteration of the lazy stage begins by selecting a sample $\boldsymbol{y}^*_{\rm rand}$ in the task space with an exploration/exploitation strategy aimed at balancing tree expansions toward unvisited regions and the desired set-point $\boldsymbol{y}^*_M$. The vertex $v_{\rm near}$, that is the nearest in $\mathcal{V}'$ to the point $\boldsymbol{y}^*_{\rm rand}$ in terms of the above-mentioned metric $d(\cdot, \boldsymbol{y}^*_{\rm rand})$, is selected, and an attempt of expanding the tree from $v_{\rm near}$ is made.

The configuration $\boldsymbol{q}_{\rm near}$ and the weights $\boldsymbol{w}_{\rm near}$ are extracted from $v_{\rm near}$; call t_k the time instant associated to $v_{\rm near}$. A random primitive is selected from U using the weights in $\boldsymbol{w}_{\rm near}$ as probabilities [2]. Let $\boldsymbol{u}^k_{\rm CoM}$ be the CoM displacement associated to this primitive, and T_k its duration. Using eq. (1), the reference pose of the CoM frame $\boldsymbol{q}_{\rm CoM}(t_{k+1})$, with $t_{k+1} = t_k + T_k$, is computed. The obtained reference pose of the CoM frame $\boldsymbol{q}^{\rm new}_{\rm CoM} = \boldsymbol{q}_{\rm CoM}(t_{k+1})$ is checked for collisions with workspace obstacles (or portions of them) contained in the planning zone $\mathcal{S}(\boldsymbol{q}_{\rm root})$ using the simplified robot model $\mathcal{B}(\boldsymbol{q}_{\rm new})$. In general, different choices of such simplified model are possible. For example, bounding volumes of either the whole robot or parts of it (e.g., its upper-part) are suitable options. Another possibility is to check collisions only at footstep level [3]. In our experiments we used a cylindrical bounding box (see Sect. V for details).

[2]We carefully avoid the selection of the `free_CoM` primitive at this stage, since it does not include any predefined trajectory of the CoM. This can be done simply by setting to zero the associated weight.

[3]This is admissible within our framework since, at each tree expansion, footstep placement automatically emerges from the swing foot trajectory specified by the selected primitive (see Sect. II).

Procedure 2: LazyStage($\bar{\boldsymbol{q}}$, ΔT_P^L)

root the tree $\mathcal{T}$ at $v_{\rm root} = (\bar{\boldsymbol{q}}, \bar{\boldsymbol{u}}_{\rm CoM}, \boldsymbol{w}_{\rm root})$;
$\mathcal{V}' \leftarrow \{v_{\rm root}\}$;
$elapsed\_time \leftarrow 0$;
repeat
  select a random sample $\boldsymbol{y}^*_{\rm rand}$ in the task space;
  select the nearest vertex $v_{\rm near}$ in $\mathcal{V}'$ to $\boldsymbol{y}^*_{\rm rand}$ according to $d(\cdot, \boldsymbol{y}^*_{\rm rand})$;
  extract $\boldsymbol{q}_{\rm near}$ and $\boldsymbol{w}_{\rm near}$ from $v_{\rm near}$, and retrieve the associated time instant t_k;
  select from U a CoM primitive $\boldsymbol{u}^k_{\rm CoM}$ with probability $\boldsymbol{w}_{\rm near}$, and retrieve the associated duration T_k;
  compute $\boldsymbol{q}^{\rm new}_{\rm CoM}$ according to $\boldsymbol{q}^{\rm near}_{\rm CoM}$ and $\boldsymbol{u}^k_{\rm CoM}$;
  if collision free **then**
    $v_{\rm new} \leftarrow ((\boldsymbol{q}^{\rm new}_{\rm CoM}, \emptyset)^T, \boldsymbol{u}^k_{\rm CoM}, \boldsymbol{w}_{\rm new})$;
    add vertex $v_{\rm new}$ in $\mathcal{T}$ as a child of $v_{\rm near}$;
  end
  update the weights in $v_{\rm near}$ as in (4-5);
  extract a set $\mathcal{V}' \subseteq \mathcal{V}$ of vertexes admissible for expansion;
  $elapsed\_time \leftarrow$ get the elapsed time;
until $\mathcal{V}' = \emptyset$ **or** $elapsed\_time > \Delta T_P^L$;
return $\mathcal{T}$;

If $\boldsymbol{q}^{\rm new}_{\rm CoM}$ results to be collision-free, a new vertex $v_{\rm new}$ is constructed as $(\boldsymbol{q}_{\rm new}, \boldsymbol{u}^k_{\rm CoM}, \boldsymbol{w}_{\rm new})$, where the sub-vector $\boldsymbol{q}^{\rm new}_{\rm jnt}$ of $\boldsymbol{q}_{\rm new}$ is left undefined and $\boldsymbol{w}_{\rm new}$ is set to $\boldsymbol{w}_{\rm init}$, and it is added to $\mathcal{T}$ as a child of $v_{\rm near}$ (such information may be stored with the vertex at the time of its creation), otherwise no vertex is added to $\mathcal{T}$.

Assume that the h-th primitive had been selected for lazy expansion, in either succeeded or failed attempt, the weights associated to $v_{\rm near}$ are updated by zeroing $w^h_{\rm near}$ and accordingly compensating the others (in such a way to avoid the choice of the corresponding primitive in a future expansion attempt, in favor of unselected primitives[4]) as

$$w^h_{\rm near} = 0 \tag{4}$$

$$w^i_{\rm near} = w^i_{\rm near} + \frac{w^h_{\rm near}}{M-1},\ i \neq h,\ w^i_{\rm near} > 0, \tag{5}$$

where $M \leq N$ is the number of positive weights in $\boldsymbol{w}_{\rm near}$ before the update.

Before starting a new iteration, the set of vertexes admissible for the next expansion attempt is constructed. At each iteration of the lazy stage, a set (possibly empty) of branches in $\mathcal{T}$ provides a set of corresponding local plans, in the sense described above. The respective ending vertexes of such branches are not allowed to be selected for expansion attempts, as expanding one of them provides no new local plan. Furthermore, also vertexes that had been already expanded through all the available primitives, i.e., vertexes in which $\boldsymbol{w} = \{0, \ldots, 0\}$, are not allowed to be selected for expansion attempts. Consequently, all the vertexes in $\mathcal{V}$, but those that either conclude a branch providing a local plan or contain only zero weights, are added to $\mathcal{V}'$.

[4]Note that the scope of such update is only local, since the same primitive can be still chosen to expand a different vertex.

B. The Validation Stage

The validation stage (see Procedure 3) starts by selecting the best local plan in the tree $\mathcal{T}$. The aim is to find a solution that allows the humanoid to approach the goal as much as possible, and possibly complete the task. The best local plan p^* is then selected as the one in $\mathcal{T}$ whose ending vertex is the nearest to the point $\boldsymbol{y}_M^*$ in terms of the compatibility metric $d(\cdot, \boldsymbol{y}_M^*)$ described above.

Let K be the number of vertexes in p^*. The validation stage proceeds by verifying the feasibility of each vertex v_k in the selected local plan (except for the root vertex $v_{\rm root}$ that is guaranteed to be feasible by construction) by scanning it from the root to the ending vertex v_K.

The n-vector of joint angles $\boldsymbol{q}_{\rm jnt}^k$ is extracted from v_k. If this is defined, the vertex had already been validated, together with the edge joining v_k to v_{k-1}, and the process proceeds by considering the next vertex along the plan. If $\boldsymbol{q}_{\rm jnt}^k$ is yet undefined, the motion generator is invoked.

The motion generator (see Procedure 4) extracts the configuration $\boldsymbol{q}_{k-1}$ from the last validated vertex v_{k-1} along the plan (that coincides with the parent vertex of v_k in $\mathcal{T}$), and retrieves the CoM primitive $\boldsymbol{u}_{\rm CoM}^{k-1}$ that has produced v_k as a child of v_{k-1} during the lazy stage. Let t_{k-1} the time instant associated to v_{k-1} and T_{k-1} the duration of the primitive $\boldsymbol{u}_{\rm CoM}^{k-1}$. Using eq. (1), the reference pose of the CoM frame $\boldsymbol{q}_{\rm CoM}(t)$ is computed, for $t \in [t_{k-1}, t_k]$, with $t_k = t_{k-1} + T_{k-1}$. This, together with the associated reference motion of the swinging foot (included in the primitive), defines the current locomotion task.

At this point, a whole-body motion is computed such that the robot starts from $\boldsymbol{q}_{k-1}$, executes the current locomotion task, eventually reaches the desired set-point $\boldsymbol{y}_M^*$ and complies with requirements R2-R4. To this end, we have used a task-priority approach:

$$\boldsymbol{v}_{\rm jnt} = \boldsymbol{J}_L^\dagger \dot{\boldsymbol{y}}_L' + \boldsymbol{P}_L (\boldsymbol{J}_M \boldsymbol{P}_L)^\dagger (\dot{\boldsymbol{y}}_M' - \boldsymbol{J}_M \boldsymbol{J}_L^\dagger \dot{\boldsymbol{y}}_L') + \boldsymbol{P}_{LM}\, \boldsymbol{v}_0. \tag{6}$$

Here, $\boldsymbol{y}_L$ is the (primary) locomotion task, $\boldsymbol{J}_L$ its Jacobian and $\boldsymbol{P}_L = \boldsymbol{I} - \boldsymbol{J}_L^\dagger \boldsymbol{J}_L$; $\boldsymbol{y}_M$ is the (secondary) manipulation task, $\boldsymbol{J}_M$ its Jacobian and $\boldsymbol{P}_{LM} = \boldsymbol{P}_L - (\boldsymbol{J}_M \boldsymbol{P}_L)^\dagger (\boldsymbol{J}_M \boldsymbol{P}_L)$. Also, we set $\dot{\boldsymbol{y}}_L' = \dot{\boldsymbol{y}}_L^* + \boldsymbol{K}_L \boldsymbol{e}_L$ and $\dot{\boldsymbol{y}}_M' = \boldsymbol{K}_M \boldsymbol{e}_M$, where $\boldsymbol{e}_L = \boldsymbol{y}_L^* - \boldsymbol{y}_L$, with $\boldsymbol{y}_L^*$ the reference value of $\boldsymbol{y}_L$, and $\boldsymbol{e}_M = \boldsymbol{y}_M^* - \boldsymbol{y}_M$. Finally, $\boldsymbol{K}_L$ and $\boldsymbol{K}_M$ are positive definite gain matrices and the null-space vector $\boldsymbol{v}_0$ in (6) is set to

$$\boldsymbol{v}_0 = -\eta \nabla_{\boldsymbol{q}_{\rm jnt}} H(\boldsymbol{q}_{\rm jnt}), \tag{7}$$

where η is a positive stepsize and $H(\boldsymbol{q}_{\rm jnt})$ is a cost function chosen so as to maximize the available joint range.

The joint trajectories generated by the kinematic control law (6) are continuously checked for collisions with workspace obstacles (or portions of them) included in the zone $\mathcal{S}(\boldsymbol{q}_{\rm root})$ and self-collisions (requirement R2) using the exact robot model $\mathcal{R}(\boldsymbol{q})$, and for violation of position/velocity joint limits (requirement R3).

If a violation occurs, motion generation is interrupted, the subtree $\mathcal{T}_k$ rooted at v_k is removed from $\mathcal{T}$, and the best local plan resulting from the updated tree is selected for a new validation attempt. In case no plan is available or the time budget ΔT_P^V is exceeded, the validation stage stops and a failure is returned to the main planner.

Procedure 3: ValidationStage($\mathcal{T}$, ΔT_P^V)

```
select the best local plan p* in T;
elapsed_time ← 0;
while p* ≠ ∅ and elapsed_time ≤ ΔT_P^V do
    k ← 0;
    repeat
        k ← k + 1;
        extract q_jnt^k from v_k;
        if q_jnt^k = ∅ then
            [q_k, q_{k-1}q_k] ← MotionGeneration(v_{k-1}, v_k);
            if q_k ≠ (∅, ∅)^T then
                update vertex v_k with q_k and add edge
                q_{k-1}q_k to T;
            else
                remove subtree T_k rooted at v_k from T;
            end
        end
    until k = K or q_k = (∅, ∅)^T;
    if k = K and q_k ≠ (∅, ∅)^T then
        retrieve whole-body motion q(t) joining v_root to v_K;
        return q(t);
    end
    select the best local plan p* in T;
    elapsed_time ← get the elapsed time;
end
return ∅;
```

If no violation occurs, the integration reaches t_k, i.e., the time instant associated to v_k, and the vertex is updated with the full configuration $\boldsymbol{q}_k$, where both the world pose $\boldsymbol{q}_{\rm CoM}^k$ of the frame attached to the CoM and the n-vector $\boldsymbol{q}_{\rm jnt}^k$ of joint angles are defined; the edge, i.e., the whole-body motion, joining v_{k-1} to v_k is also added to the tree $\mathcal{T}$, and the process proceeds by considering the next vertex along the plan.

If the ending vertex v_K contains a configuration $\boldsymbol{q}_K$ that is closer than the threshold $\bar{d}$ to the desired set-point $\boldsymbol{y}_M^*$, i.e., $d(\boldsymbol{q}_K, \boldsymbol{y}_M^*) < \bar{d}$, and the desired set-point is not yet achieved, i.e., $\boldsymbol{f}(\boldsymbol{q}_K) \neq \boldsymbol{y}_M^*$, the validation stage performs a last operation (not explicitly indicated in Procedure 3 for sake of illustration), otherwise the whole-body motion joining $v_{\rm root}$ to v_K is returned. Such operation consists in trying to conclude the local plan with the `free_CoM` primitive for allowing the robot to complete the assigned task. This primitive leaves the CoM free to move, constrains the feet to remain fixed and its duration is determined by the time needed to reach the desired set-point. In order to generate the motion, kinematic control law (6) is still used, starting the integration at $\boldsymbol{q}_K$ and explicitly checking static equilibrium. In case of failure, the search of a valid plan continues as mentioned above, otherwise the whole-body motion $\boldsymbol{q}(t)$ starting from $v_{\rm root}$ and allowing the robot to achieve the desired set-point is returned.

V. Planning Experiments

The proposed anytime framework has been implemented in V-REP on an Intel Core i7 running at 2.70 GHz. The

Procedure 4: MotionGeneration(v_{k-1}, v_k)

```
extract q_{k-1} from v_{k-1}, and retrieve the associated time
  instant t_{k-1};
extract u_CoM^{k-1} from v_k, and retrieve the associated duration
  T_{k-1};
compute current locomotion and manipulation tasks;
repeat
    generate motion by integrating joint velocities (6);
    if collision or joint position/velocity limit violation then
        return [∅, ∅];
    end
until t = t_{k-1} + T_{k-1};
return [q_k, q_{k-1}q_k];
```

chosen robotic platform is the NAO small humanoid, with the right hand as end-effector. The planning zone consists of a sphere with radius equal to 1.25 m. The predefined time budget $\overline{\Delta T}_P$ has been set to 5 s, and we split the time budget ΔT_P between the two stages as $\Delta T_P^L = 3.5$ s and $\Delta T_P^V = \Delta T_P - \Delta T_P^L$. The threshold $\bar{d}$ used in the lazy stage has been set to 0.15 m, while the initial weights are simply set to $\boldsymbol{w}_{\text{init}} = \{\frac{1}{N}, \ldots, \frac{1}{N}\}$. Finally, for the simplified robot model $\mathcal{B}(\cdot)$, we have used a fixed size cylindrical bounding box (with radius and height, respectively, equal to 0.16 m and 0.56 m) that completely includes the humanoid at its initial configuration. Note that such choice is particularly suited for tackling scenarios (as those considered in the following) where the robot moves on a flat ground. However, considering situations where the robot needs to step over and onto low objects is also possible with our planner simply by performing collision checking at footsteps level during the lazy stage, instead of using bounding volumes.

The set of CoM primitives is defined as $U = \{\texttt{free\_CoM} \cup U_{\text{CoM}}^D\}$. Here, `free_CoM` is a *non-stepping* primitive that allows the CoM to move freely, as long as both feet remain fixed; U_{CoM}^D is a subset of *dynamic* steps extracted from various types of gaits precomputed by the intrinsically stable MPC framework presented in [19]. In particular, U_{CoM}^D includes forward, backward, curved and diagonal steps.

In order to avoid unnatural motions, we have decided to not constrain the hand motion during the early stages of the planning. This is obtained by activating the manipulation task only when the hand is within a certain sphere centered at the desired set-point $\boldsymbol{y}_M^*$. In such case, the motion is generated according to eq. (6). In the other case, the secondary task is removed from eq. (6). Note that, even in this condition, the planning process is biased towards the desired set-point thanks to the use of our compatibility metric for both the selection of vertexes to be expanded and local plans to be validated in, respectively, the lazy and the validation stages.

We consider two planning scenarios of increasing complexity. In both of them, the robot is assigned the task of grasping a ball placed on a table that is outside its workspace at the initial configuration.

In the first scenario (Fig. 2), a wall is obstructing the path from the robot to the destination. For the first 5 s, the robot is allowed to plan without moving. In this phase, large portion

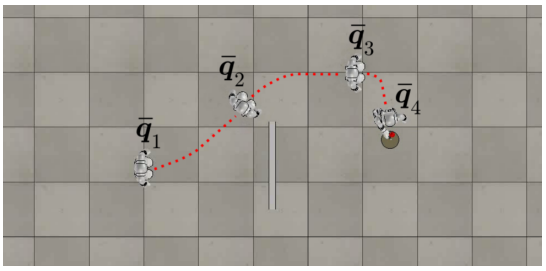

Fig. 2. Planning scenario 1: snapshots from a solution. $\bar{\boldsymbol{q}}_i$: configuration assumed by the robot at the beginning of the i-th planning interval.

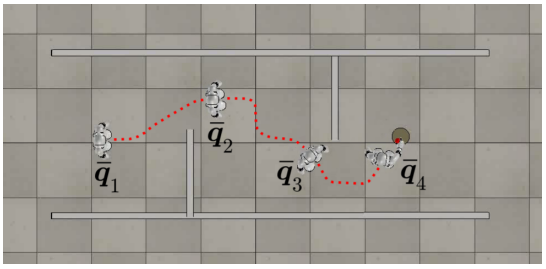

Fig. 3. Planning scenario 2: snapshots from a solution. $\bar{\boldsymbol{q}}_i$: configuration assumed by the robot at the beginning of the i-th planning interval.

of the wall is contained within the initial planning zone, and consequently the planner generates a sequence of dynamic steps in the forward direction followed by curved steps in the left direction allowing the robot to avoid the wall. Then, while the robot executes these steps, it plans its future motion that will start at $\bar{\boldsymbol{q}}_2$. This process is repeated and the result is a sequence of steps that allows the robot to approach the desired set-point, correctly completing the task.

The second scenario (Fig. 3) constrains the robot to move only within a corridor in which two walls create two consecutive narrow passages. At the beginning, the robot plans a solution allowing it to manage the first narrow passage. During the execution of such plan, the robot plans a new solution for managing the second narrow passage. The task is then successfully completed.

Table I collects some data related to the planner performance in the two scenarios. Both experiments have required three planning intervals. The table reports: the time budget for each invocation of the LMP, the time needed for planning a solution within each invocation, the number of lazy plans and the duration of the generated motion. Note that our LMP always produces a solution before the deliberation time runs out (i.e., the planning time is smaller than the time budget). In general, when the LMP is not able to solve the local planning problem within the time budget, the robot performs a safe stopping motion[5] at the end of the current plan and then invokes again the LMP with the initial time budget $\overline{\Delta T}_P$.

We encourage the reader to see the accompanying video to better appreciate the effectiveness of the generated motions.

We do not perform any explicit comparison with other methods in the literature because they either propose off-line

[5]The generation of the stopping motion is out of the scope of the paper and its description is omitted for this reason.

i	time budget (s)	planning time (s)	# lazy plans	motion duration (s)
Experiment 1				
0	5	4.95	28	26.9
1	26.9	7.04	36	26.4
2	26.4	6.07	51	15.1
Experiment 2				
0	5	4.85	3	29.1
1	29.1	6.94	10	27.4
2	27.4	5.43	31	22.6

TABLE I

PLANNER PERFORMANCE DATA.

i	time budget (s)	planning time (s)	# lazy plans	motion duration (s)
0	5	4.17	41	26.9
1	26.9	4.71	3	28.8
2	28.8	6.52	9	20.7
3	20.7	10.25	2	75.0
4	75.0	6.49	17	21.0
5	21.0	7.26	18	31.5
6	31.5	7.07	17	26.4
7	26.4	7.95	12	28.3

TABLE II

PLANNER PERFORMANCE DATA FOR THE DEADLOCK SCENARIO.

approaches (e.g., [9]) or they are not easily implementable for generating humanoid whole-body motions (e.g., [10], [14]). Just as example, the planner in [9] needs 104 s for computing a plan in the first scenario, forcing the robot to wait for more than one minute before starting moving (as opposed to the 4.95 s of this approach). Similar results were obtained for the other scenarios. This confirms the validity of the proposed approach.

VI. DEADLOCK MANAGEMENT

As in Sect. III, the LMP, at each invocation, provides a local plan that is guaranteed to be feasible within a limited task space area, i.e., the planning zone, and allows the robot to approach as much as possible the desired set-point $\boldsymbol{y}_M^*$.

Until now, we have considered that the planning zone has always a fixed geometry (e.g., a sphere of a given radius), while its location is determined by the last configuration that the humanoid will assume at the end of the simultaneously executed local plan. Such strategy provides the planner only with very local information about the environment. In principle, consecutive invocations of the LMP might generate movements that enforce the robot to repeatedly navigate among same regions of the task space. This behavior causes full-fledged deadlock situations.

Such problem can be easily eliminated by allowing the planner to keep memory of previously considered planning zones. In fact, this can be done by appropriately instantiating the planning zone as the union of the local one, described in Sect. III, and all the previously considered (see Fig. 4).

To exhibit how deadlocks can be avoided by keeping memory of previously considered planning zones, we compared the two versions of our algorithm (*without memory* and *with memory*) in a deadlock-prone scenario where a concave obstacle, constituted by three walls, creates a local minima. Note that, in the considered scenario, the concave obstacle is not entirely contained in the initial planning zone.

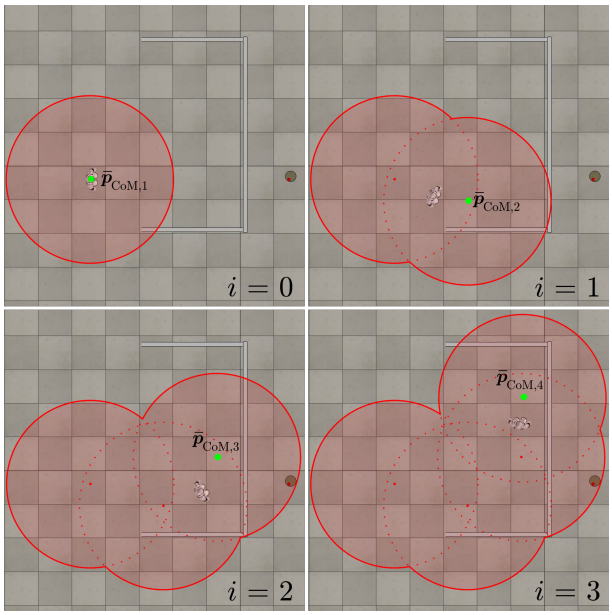

Fig. 4. Planning with memory: at the i-th planning interval, the planning zone is defined as the union of the local sphere-shaped planning zone, centered at the position of the CoM $\bar{\boldsymbol{p}}_{\mathrm{CoM,i+1}}$ (shown in green) that the robot will have at the time instant t_{i+1}, and all the previous ones.

Using the version without memory, the robot moves toward the desired set-point until it reaches the proximity of the corner formed by two walls; at this point the LMP, due to collisions with the walls, can generate only curved steps in the left direction. Then, backward movements are generated, since they represent the plan allowing the robot to approach the goal as much as possible. Such movements bring the robot in already visited zones of the workspace, from which the LMP generates forward movements to approach the boundary of the current planning zone. As before, backward movements are generated, bringing again the robot in the closed corner; this loop continues, trapping the robot within the concave obstacle, and precluding it to fulfill the task.

The version with memory results effective. A solution to the planning problem is shown in Fig. 5, while the same performance indexes of Table I are reported in Table II for this scenario. The robot proceeds toward the goal until it reaches the proximity of the wall obstructing the passage. At the third invocation, the LMP, taking into account the planning zone consisting in the union of the local one and the two previous ones, produces a solution that moves the robot outside the concave obstacle. It is important to emphasize that this behavior results automatically from the candidate local plan chosen in the validation stage. Sequences of dynamic steps are then produced, allowing the robot to approach the goal and finally to complete the task.

We encourage the reader to watch the accompanying video to better appreciate the comparison of the two versions.

VII. TOWARDS A SENSOR-BASED PLANNER

The proposed framework assumes that, at each replanning step, only a limited portion of the environment is available

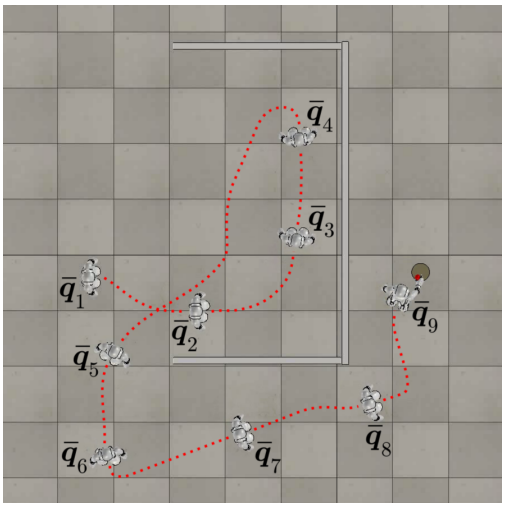

Fig. 5. Planning scenario 3: snapshots from a solution. The first four planning zones are the ones shown in Fig. 4. Already at the third planning interval, the robot is able to plan a motion that gets itself out from the concave obstacle, allowing the robot to fulfill the assigned task.

to the planner. This allows to avoid the off-line computation of a complete solution that, being in general expensive, may enforce the robot to wait for a long time before start moving.

The structure of our framework makes it particularly suited for a sensor-based extension. In fact, a sensor typically provides local information about the environment and this particularly fits with the definition of the planning zone given in Sect. III, since it acts as a map provided by an ideal sensor, that can be easily replaced by a map incrementally constructed by the robot according to information gathered by an on-board sensor (e.g., a camera on the robot head).

The interleaved procedure can be appropriately modified such that the LMP is invoked after a given portion of the current local plan is executed, then using the updated map for computing a solution for the next execution interval, while simultaneously executing the remaining part of the current local plan. A sensor-based extension of the planner can identify candidate local plans as branches of the constructed tree that bring the robot outside the frontiers of the current map, as analogously done by the presented anytime planner with the planning zone. The planner extension to the sensor-based case is the subject of our current work.

VIII. Conclusions

We have described an anytime planning/replanning scheme that allows a humanoid to accomplish an assigned task that implicitly requires stepping. The algorithm interleaves planning and execution: a previously planned whole-body motion is executed while a new solution for the subsequent execution is simultaneously planned by means of a randomized local planner. Such a planner works in two stages: a first stage quickly expands a tree in the configuration-time space, performing lazy collision checks; then, a second stage searches the tree for a feasible, collision-free whole-body motion providing a solution for the local planning problem. Finally, we discussed how the planner can avoid deadlock situations and sketched a possible extension to the sensor-based case. The proposed method has been implemented in V-REP for the NAO robot and successfully tested in increasing complexity scenarios.

In the future, we will investigate the discussed sensor-based extension, to perform real-robot experiments. Moreover, future works will focus on adding novel CoM movements, in order to allow the planner to fulfill tasks that require more complex motions (e.g., crouching). Finally, it would be interesting to include a second-order motion generator, that allows to take into account torque bounds.

References

[1] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," *Robotics Research*, pp. 365–374, 2005.

[2] J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins, and T. Kanade, "Footstep planning for the honda asimo humanoid," in *2005 IEEE Int. Conf. on Robotics and Automation*, 2005, pp. 629–634.

[3] E. Yoshida, I. Belousov, C. Esteves, and J.-P. Laumond, "Humanoid motion planning for dynamic tasks," in *2005 IEEE-RAS Int. Conf. on Humanoid Robots*, 2005, pp. 1–6.

[4] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[5] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond, "Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1089–1103, 2013.

[6] K. Bouyarmane and A. Kheddar, "Humanoid robot locomotion and manipulation step planning," *Advanced Robotics*, vol. 26, no. 10, pp. 1099–1126, 2012.

[7] M. Cognetti, P. Mohammadi, and G. Oriolo, "Whole-body motion planning for humanoids based on com movement primitives," in *2015 IEEE-RAS Int. Conf. on Humanoid Robots*, 2015, pp. 1090–1095.

[8] M. Cognetti, V. Fioretti, and G. Oriolo, "Whole-body planning for humanoids along deformable tasks," in *2016 IEEE Int. Conf. on Robotics and Automation*, 2016, pp. 1615–1620.

[9] P. Ferrari, M. Cognetti, and G. Oriolo, "Humanoid whole-body planning for loco-manipulation tasks," in *2017 IEEE Int. Conf. on Robotics and Automation*, 2017, pp. 4741–4746.

[10] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, 2004, pp. 767–774.

[11] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm." in *ICAPS*, 2005, pp. 262–271.

[12] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *2012 IEEE-RAS Int. Conf. on Humanoid Robots*, 2012, pp. 674–679.

[13] D. Ferguson and A. Stentz, "Anytime rrts," in *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 5369–5375.

[14] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *2011 IEEE Int. Conf. on Robotics and Automation*, 2011, pp. 1478–1483.

[15] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *2000 IEEE Int. Conf. on Robotics and Automation*, vol. 1, 2000, pp. 521–528.

[16] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *The International Journal of Robotics Research*, pp. 403–417, 2003.

[17] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *2015 IEEE Int. Conf. on Robotics and Automation*, 2015, pp. 2951–2957.

[18] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Efficient motion planning for humanoid robots using lazy collision checking and enlarged robot models," in *2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007, pp. 3062–3067.

[19] N. Scianca, M. Cognetti, D. De Simone, L. Lanari, and G. Oriolo, "Intrinsically stable MPC for humanoid gait generation," in *16th IEEE-RAS Int. Conf. on Humanoid Robots*, 2016, pp. 101–108.